# Introduction

## Artificial Intelligence (AI)

Study of using computers to solve problems that involve percetron/intelligence.

Neural networks $\subseteq$ AI

## What is a neural network?

— Neural networks are composed of processing elements and connections. Each processing element has a single output signal that fans out along connections to each other processing elements.

## A neural network is characterizes by

(1) Its pattern of connection between the neurons (called is architecture).

(2) Its method of determining the weights on the connections (called its training or learning).

(3) Its activation function.

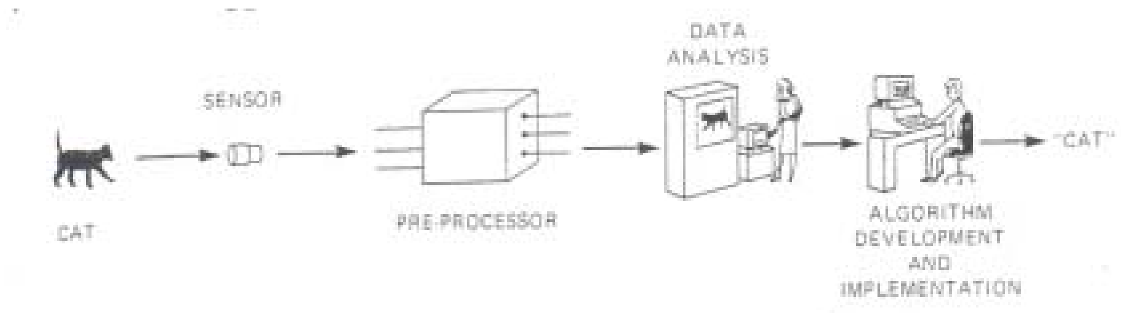
Neural networks are not programmed, they learn by example.

Traditional methods: develop computer programs.

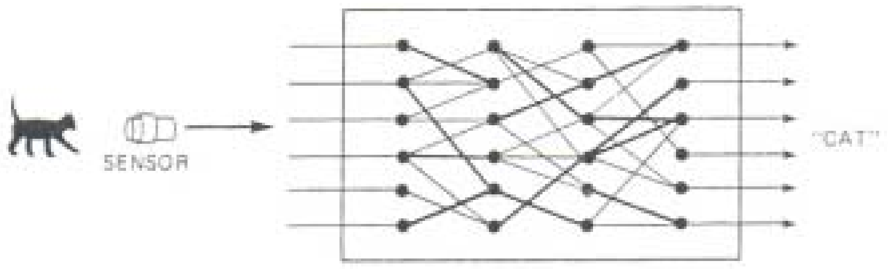
Neural networks: begin with sample inputs and outputs, and learn to provide the correct outputs for each input.

Example:

Traditional approach



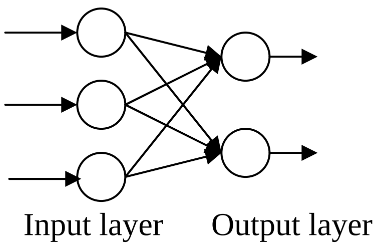
Neural networks approach



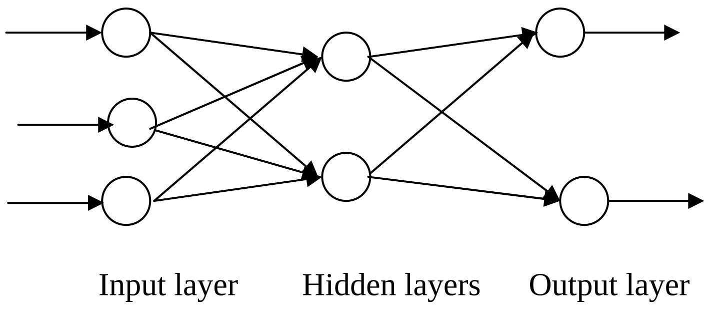
## How are neural network used?

## Typical architecture

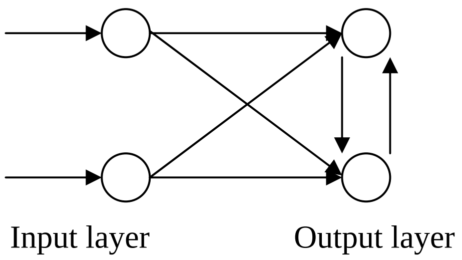
A single-layer neural net



A multi-layer net



A recurrent net



## Supervised learning

A set of training vectors with a corresponding set of desired output vectors is trained to adjust the weights in a neural net.

## Unsupervised learning

A sequence of input vectors is provided, but no target vectors are specified.

## Common activation functions

(1) Identify function: $f(x) = x \quad \textit{for all } x.$

(2) Binary step function (with threshold $\theta$)

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

(3) Sigmoid function:

$$f(x) = \frac{1}{1+e^{-\sigma x}} \longrightarrow \quad \sigma = 1, \quad f(x) = \frac{1}{1+e^{-x}} \qquad [0, 1]$$

Bipolar sigmoid

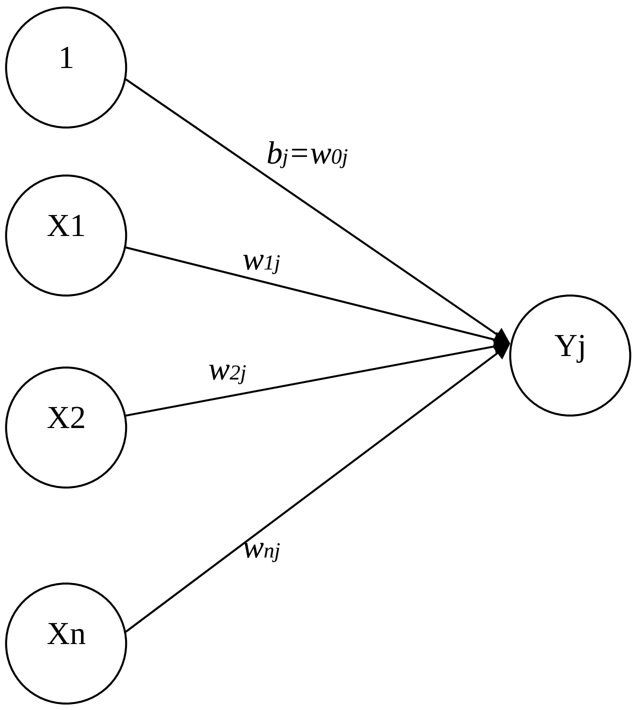
$$g(x) = 2f(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}} \qquad [-1, 1]$$

## The perceptron

— The perceptron learns to classify patterns through supervised learning.

—The patterns it classifies are usually binary-valued vectors, and the classification categories are expressed as binary vectors.

—The percetron is limited to two layers of processing units with a single layer of adaptive weights between then.



Input vector $X = \{1, x_1, x_2, x_3, .....x_n\}$

Weight $W = \{w_{oj}, w_{1j}, w_{2j}, ..., w_{nj}\}$

$$net_j = XW_j = \sum_{i=0}^{n} x_i w_{ij} = w_{0j} + \sum_{i=1}^{n} x_i w_{ij}$$

$$= b_j + \sum_{i=1}^{n} x_i w_{ij}$$

$$Y_j = \begin{cases} 1 & if \quad net_j > 0 \\ 0 & otherwise \end{cases}$$

Using the simplest perceptron learning rule

$$w_{ij}^{new} = w_{ij}^{old} + \alpha(t_j - y_j)x_i$$

where

$x_i = 1$ or 0, the value of input $i$

$y_j =$ the output value procedured by output unit $j$

$t_j =$ the target value for output $j$.

$\alpha =$ a constant (" the learning rate")

## Training rule

If output equal to input, then do nothing (no change weights)

If output not equal to input, then $w_{ij}^{new} = w_{ij}^{old} + \alpha(t_j - y_j)x_i$

$$Decision\ \alpha\ is\ art$$

## Training

Performance can be measured by a root-mean-squares error (RMSE) value.

$$RMSE = \sqrt{\frac{\sum_p \sum_j (t_{jp} - y_{jp})^2}{n_p n_o}}$$

where

$n_p$ : *number of patterns in the training set.*

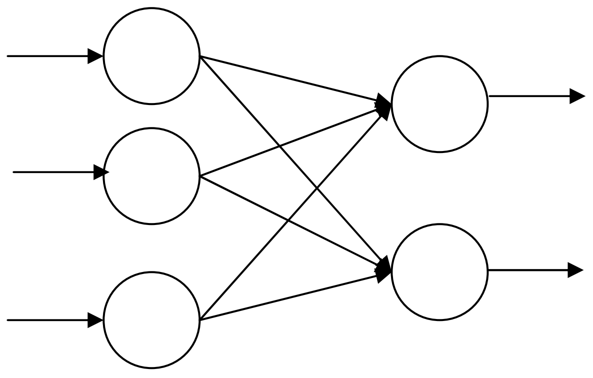
$n_o$ : *numnber of units in the output layer.*

$t_{jp}$ : *the target value for output unit j after presentation of pattern p.*

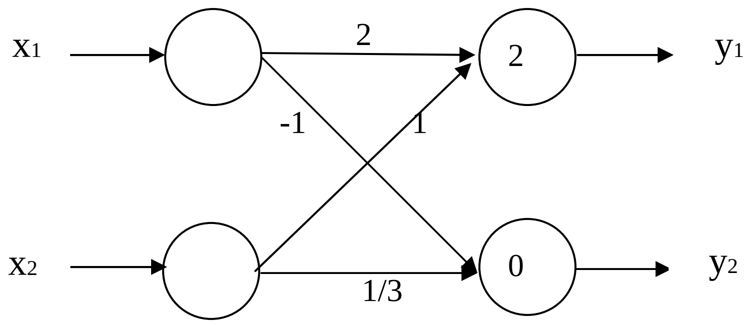
$y_{jp}$ : *the output value proceduced by output unit j after presentation of pattern p.*

## <u>Percetron convergence theorem</u>

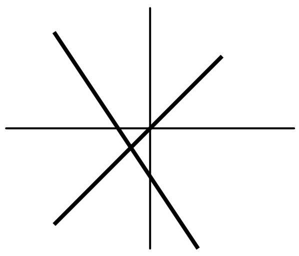
If there is of weights that correctly classify the (linearly separable) training patterns, then the learning algorithm will find one such set in a finite number of iterations.



Example (座標旋轉)



$$2x_1 - x_2 + 2 = 0$$
$$\therefore\ x_2 = -2x_1 - 2$$
$$-x_1 + \frac{1}{3}x_2 + 0 = 0$$
$$\therefore x_1 = \frac{1}{3}x_2$$



Percetron can learn linearly separable classes.

Linear separable: Regions or classes which can be partitioned into two spaces using linear boundaries such as a line, plane or hyperplane give by $XW$ , are said to linearly separable.

Linearly separable                    not linearly separable

<u>Learning procedure</u>

Choose initial weights $w_0, w_1, w_2$

Iterate until stopping criteria reaches

Choose sample *X* with desired output *t*

Compute $Y = w_0 + \sum_i x_i w_i$

If *Y*=*t*, do nothing

Else update weights using $w_i^{new} = w_i^{old} + \alpha(t_i - Y_i)X$

End iterate

For linearly separable tasks, the procetron algorithm will always find a set of weights to do the job.

For example, two classes C, K

| C | K |
|---|---|
| (1,2) | (3,3) |
| (1,1) | (3,2) |
| (-1,0) | (4,3) |

*If* $X \in C$, *then we want* $t = 1$;
*If* $X \in K$, *then we want* $t = 0$.

Activation function

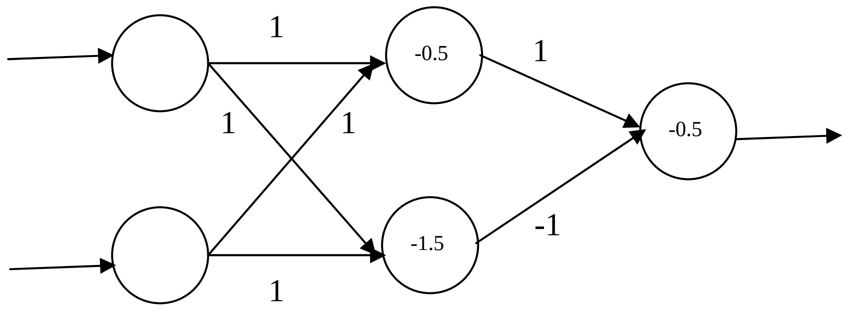
$$f(net) = \begin{cases} 1 & if\ net > 0 \\ 0 & otherwise \end{cases}$$

The chief limitation of the percetron is that output unit can classify only linearly separable patterns.

Exclusive OR (XOR)

| Truth table | | |
|---|---|---|
| $x_1$ | $x_2$ | $Y$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

The percetron is incapable of solving XOR problem.

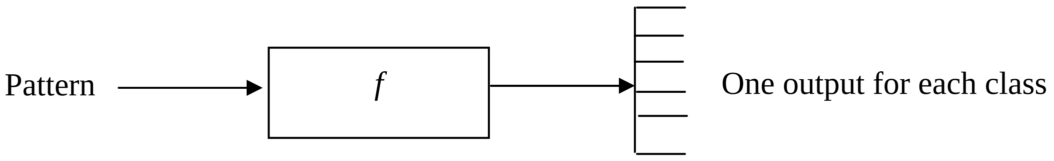


# Multilayer Feedforward Networks

## Multilayer feedforward (MLFF) Networks

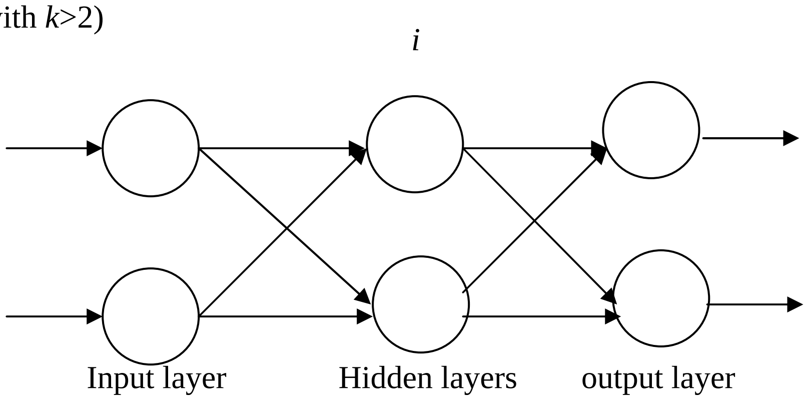
## (Multilayer procetron)

—Very useful for function estimation



—Usually can be divided into layers

(*k* layers with *k*>2)



— Activation function of unit *i* in layer L>1

$$y = \varphi_i(\sum_{j=1}^{n} w_{ij} a_j + b_i)$$

where $a_j$ is the *j*th activation value in layer L-1;

$w_{ij}$ is the weight connecting unit *i* in layer L to unit *j* in layer L-1;

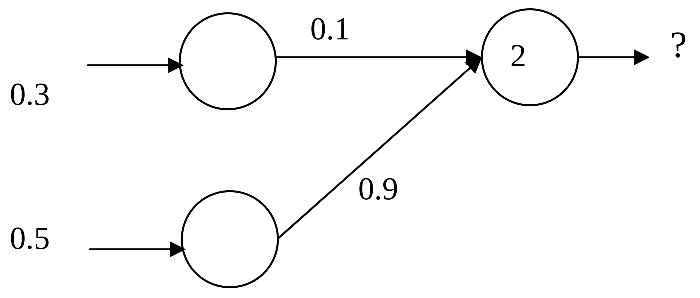
$b_i$ is a bias asscoated with unit *i*;

$\varphi_i$ is a nonlinear function

— Usually $\varphi_i$ is the same at every node, so we just call it $\varphi$.

Usually $\varphi(x) = \dfrac{1}{1+e^{-x}}$ (call sigmoid or logistics function)

Example



$$net = (0.3 * 0.1) + (0.5 * 0.9) + 2 = 2.48$$

$$y = \frac{1}{1+e^{-net}} = \frac{1}{1+e^{-2.48}} = 0.923$$

# Back-propagation Learning

## Back-propagation algorithm

—An algorithm for learning the appropriate weights of a multiplayer feedforward netwoks (generalized $\delta$-rule).

—Supervised learning

$\Longrightarrow$ You must provide the network with input/output pairs during training.

—Activation function for a non-input unit $i$ is

$$a_i = f_i\Big(\sum_{\substack{j\in previous\\ layer}} w_{ij}a_j + b_i\Big)$$

Usually, $f_i$ does not depend on $i$, $f_i = f$ for every $i$.

Usually, $f(net) = \dfrac{1}{1+e^{-net}} + c,$ where $c$ is a constant, e.g., $c = 0$ or $c = -0.5$.

—The back-propagation algorithm involves a forward pass and a backward pass.

$1^o$ Forward pass:

—The net input to node $i$ for pattern $p$ is

$$net_{pi} = \sum_{\substack{j\in previous\\ \text{layer}}} w_{ij}a_{pj} + b_i$$

where $a_{pj}$ is the activation value of unit $j$ for pattern $p$;
$w_{ij}$ is the weight form unit $j$ to unit $i$;
$b_i$ is a bias associated with unit $i$.

—The output of unit $i$ for pattern $p$ is:

$$a_{pi} = \frac{1}{1+e^{-net_{pi}}}$$

## $2^o$ Backwrd pass:

Based on gradient descent or the least squares error function

$$E = \sum_p E_p \qquad\qquad E_p = \frac{1}{2}\left\| t_p - o_p \right\|_2^2$$

where $t_p$ is the target output for the $p$th pattern.
$o_p$ is the actual output for the $p$th pattern.

—The generalized delta rule ( $\delta$ -rule)

$$w_{ij}^{new} = w_{ij}^{old} + \Delta_p w_{ij}$$

$$\Delta_p w_{ij} = \eta \delta_{pi} a_{pj}$$

where $\eta$ is the learning rate and

$$\delta_{pi} = \begin{cases} (t_{pi} - o_{pi}) f'(net_{pi}) & \text{if unit } i \text{ is an output unit.} \\ f'(net_{pi}) \sum_{\substack{k \in next \\ layer}} \delta_{pk} w_{ki} & \text{if unit } i \text{ is a hidden unit.} \end{cases}$$

—Iterative procedure:

Can update the weights after each pattern is presented or after many patterns are presented.

# Derivation of back-propagation learning algorithm

Given a training set in the form of input-output pair $(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)$,

define the error function for pattern $p$ to be

$$E_P = \frac{1}{2}\left\| t_p - o_p \right\|_2^2 = \frac{1}{2} \sum_{\substack{i \in output \\ layer}} (t_{pi} - o_{pi})^2$$

where $t_{pi}$ is the target for the $i$th output unit when $p$th pattern is presented.

$o_{pi}$ is the actual output for the $i$th output when pattern $p$ is presented.

$E = f(W, X, T)$

$X$ and $T$ are given, we want to find $W$.

—Back-propagation uses gradient descent on $E$ with respect to weights $W$

$\Longrightarrow$ Change in weights is proportional to the negative of the gradient of $E$.

$$\Delta w_{ij} = -C \frac{\partial E}{\partial w_{ij}} \qquad \text{where } C > 0 \text{ (constant)}$$

$$\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{s} \frac{\partial E_P}{\partial w_{ij}}$$

$$\frac{-\partial E_p}{\partial w_{ij}} = \frac{-\partial E_p}{\partial net_{pi}} \cdot \frac{\partial net_{pi}}{\partial w_{ij}}$$

$$\frac{\partial net_{pi}}{\partial w_{ij}} \frac{\partial}{\partial w_{ij}} \Big( \sum_{\substack{k \in previous \\ layer}} w_{ik} a_{pk} + b_i \Big) = a_{pj}$$

Let $\delta_{pi} = \dfrac{-\partial E_P}{\partial net_{pi}}$, then $\delta_{pi} = \dfrac{-\partial E_P}{\partial a_{pi}} \cdot \dfrac{\partial a_{pi}}{\partial net_{pi}}$ $\left( a_{pi} = \dfrac{1}{1 + e^{-net_{pi}}} = f(net_{pi}) \right)$

$$= \frac{-\partial E_P}{\partial a_{pi}} \cdot f'(net_{pi}) \qquad \therefore \frac{\partial a_{pi}}{\partial net_{pi}} = f'(net_{pi})$$

(1) if unit $i$ is an output unit

$$\frac{-\partial E_P}{\partial a_{pi}} = (t_{pi} - o_{pi}) \qquad \left( \because a_{pi} = o_{pi} \text{ ,and } E_P = \frac{1}{2} \sum_{\substack{i \in output \\ \text{layer}}} (t_{pi} - o_{pi})^2 \right)$$

$$\begin{aligned} \Delta_p w_{ij} &= \eta \delta_{pi} a_{pj} \\ &= \eta (t_{pi} - o_{pi}) f'(net_{pi}) \end{aligned}$$

(2) if unit $i$ is a hidden unit

$$\begin{aligned} \frac{-\partial E_P}{\partial a_{pi}} &= -\sum_{\substack{k \in next \\ layer}} \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial net_{pk}}{\partial a_{pi}} \\ &= -\sum_{\substack{k \in next \\ layer}} \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial}{\partial a_{pi}} \left( \sum_{\substack{j \in previous \\ layer}} w_{kj} a_{pj} \right) \\ &= -\sum_{\substack{\text{k} \in \text{next} \\ layer}} \frac{\partial E_p}{\partial net_{pk}} \cdot w_{ki} \\ &= \sum_{\substack{k \in next \\ later}} \delta_{pk} \cdot w_{ki} \end{aligned}$$

$$\because \delta_{pi} = \Big( \sum_{\substack{\text{k} \in \text{next} \\ \text{layer}}} \delta_{pk} w_{ki} \Big) \cdot f'(net_{pi})$$

## A numerical example of the back-propagation procedure

The initial biases and weights generated by computer for network 2-4-4 (Figure 1) are listed in Tables 1 and 2. When the first pattern, $(\overrightarrow{x_1}, \overrightarrow{t_1})$, is presented, when $\overrightarrow{x_1} = (0.017322, 1.480488)^t$ and $\overrightarrow{t_1} = (0.494200, 0.495051, 0.494171, 0.501720)^t$, the following computations illustrate the back-propagation procedure:

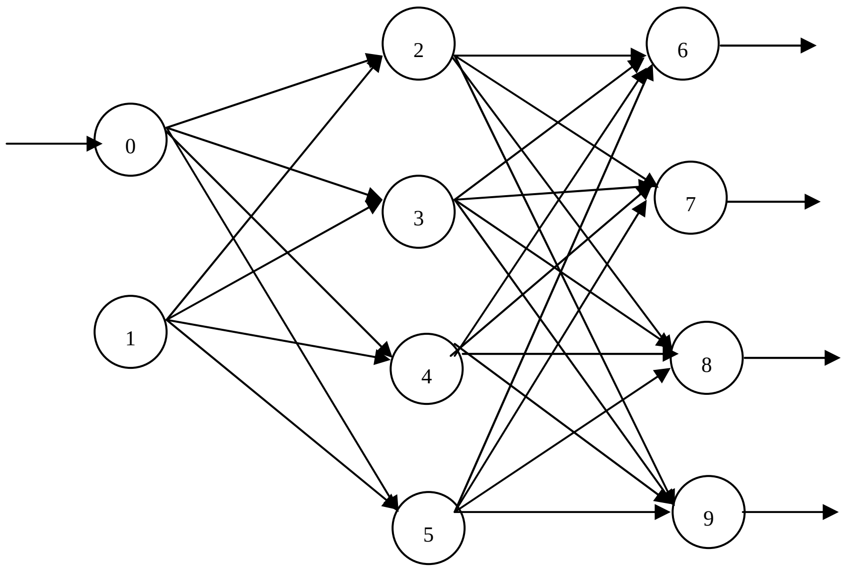


Figure 1 The architecture of network 2-4-4

Table 1 The initial random biases of network 2-4-4

| $i$ | $b_i$ | $i$ | $b_i$ |
|---|---|---|---|
| 2 | -0.444700 | 6 | 0.094012 |
| 3 | 0.410733 | 7 | -0.058550 |
| 4 | 0.358089 | 8 | -0.055376 |
| 5 | -0.005783 | 9 | -0.158925 |

Table 2 The initial random weights of network 2-4-4

| $j-i$ | $wij$ | $j-i$ | $wij$ |
|---|---|---|---|
| 0-2 | 0.121845 | 2-7 | 0.326563 |
| 1-2 | -0.384945 | 3-7 | 0.360866 |
| 0-3 | 0.474700 | 4-7 | 0.046312 |
| 1-3 | 0.131458 | 5-7 | 0.323694 |
| 0-4 | 0.194113 | 2-8 | -0.106006 |
| 1-4 | 0.187948 | 3-8 | 0.264275 |
| 0-5 | 0.318567 | 4-8 | -0.455687 |
| 1-5 | 0.117237 | 5-8 | 0.128620 |
| 2-6 | 0.069170 | 2-9 | -0.189261 |
| 3-6 | -0.088916 | 3-9 | -0.165883 |
| 4-6 | -0.432951 | 4-9 | -0.068896 |
| 5-6 | -0.270775 | 5-9 | -0.490692 |

1. Forward pass:

$$net_{p2} = (0.017322)*(0.121845)+(1.480488)*(-0.384945)-0.444700=-1.012496$$
$$net_{p3} = (0.017322)*(0.474700)+(1.480488)*(0.131458)+0.410733=0.613578$$
$$net_{p4} = (0.017322)*(0.194113)+(1.480488)*(0.187948)+0.358089=0.639706$$
$$net_{p5} = (0.017322)*(0.318567)+(1.480488)*(0.117237)-0.005783=0.173303$$
$$a_{p2} = \frac{1}{1+e^{-net_{p2}}} = \frac{1}{1+e^{1.012496}} = 0.266492$$
$$a_{p3} = \frac{1}{1+e^{-net_{p3}}} = \frac{1}{1+e^{-0.613578}} = 0.648756$$
$$a_{p4} = \frac{1}{1+e^{-net_{p4}}} = \frac{1}{1+e^{-0.639706}} = 0.654687$$
$$a_{p5} = \frac{1}{1+e^{-net_{p5}}} = \frac{1}{1+e^{-0.173303}} = 0.543218$$
$$\begin{aligned} net_{p6} &= (0.266492)*(0.069170)+(0.648756)*(-0.088916)+(0.645687)*(-0.432951) \\ &\quad +(0.543218)*(-0.270775)+0.094012=-0.375777 \end{aligned}$$
$$\begin{aligned} net_{p7} &= (0.266492)*(0.326563)+(0.648756)*(0.360866)+(0.645687)*(0.046312) \\ &\quad +(0.543218)*(0.323694)-0.058550=0.468747 \end{aligned}$$
$$\begin{aligned} net_{p8} &= (0.266492)*(-0.106006)+(0.648756)*(0.264275)+(0.654687)*(-0.455687) \\ &\quad +(0.543218)*(0.128620)\text{ - }0.055376=\text{-}0.140639 \end{aligned}$$
$$\begin{aligned} net_{p9} &= (0.266492)*(-0.189261)+(0.648756)*(-0.165883)+(0.658756)*(-0.068896) \\ &\quad +(0.543218)*(\text{-}0.490692)\text{ - }0.158925=\text{-}0.628637 \end{aligned}$$

$$a_{p6} = \frac{1}{1+e^{-net_{p6}}} = \frac{1}{1+e^{0.375777}} = 0.407146$$
$$a_{p7} = \frac{1}{1+e^{-net_{p7}}} = \frac{1}{1+e^{-0.468747}} = 0.615087$$
$$a_{p8} = \frac{1}{1+e^{-net_{p8}}} = \frac{1}{1+e^{0.140639}} = 0.464898$$
$$a_{p9} = \frac{1}{1+e^{-net_{p9}}} = \frac{1}{1+e^{0.628637}} = 0.347820$$

2. Backward pass:

2.1 If unit $i$ is an output unit

$$\delta_{pi} = (t_{pi} - o_{pi}) f'(net_{pi}), \qquad f'(net_{pi}) = f(net_{pi})[1 - f(net_{pi})] = a_{pi} * (1 - a_{pi})$$

$$\delta_{p6} = (0.494200 - 0.407146) * (0.407146)(1 - 0.407146) = 0.021013$$
$$\delta_{p7} = (0.495051 - 0.615087) * (0.615087)(1 - 0.615087) = -0.028419$$
$$\delta_{p8} = (0.494171 - 0.464898) * (0.464898)(1 - 0.464898) = 0.007282$$
$$\delta_{p9} = (0.501720 - 0.347820) * (0.347820)(1 - 0.347820) = 0.034911$$

2.2 If unit $i$ is a hidden unit

$$\delta_{pi} = f'(net_{pi}) \sum_{\substack{k \in next \\ \text{layer}}} \delta_{pk} w_{ki}, \quad f(net_{pi})[(1 - f(net_{pi})] \sum_{\substack{k \in next \\ layer}} \delta_{pk} w_{ki}$$

$$\delta_{p2} = (0.266492)(1 - 0.266492) * [(0.021013)(0.061970) + (-0.028419)(0.326563) + (0.007282)(\text{-}0.106006) + (0.034911) * (\text{-}0.189261)] = \text{-}0.002972$$
$$\delta_{\text{p}3} = (0.648756)(1 - 0.648756) * [(0.021013)(-0.088916) + (0.028419)(0.360866) + (0.007282)(0.264275) + (0.034911)(-0.165883)] = -0.003644$$
$$\delta_{p4} = (0.654687)(1 - 0.654687) * [(0.021013)(-0.432951) + (-0.028419)(0.046312) + (0.007282)(-0.455687) + (0.034911)(-0.068896) = -0.003648$$
$$\delta_{p5} = (0.543218)(1 - 0.543218) * [(0.021013)(-0.270775) + (-0.028419)(0.323694) + (0.007282)(0.128620) + (0.034911)(-0.490692) = -0.007713$$

3. Updates of weights and biases (Let $\eta = 0.2$)

$$\Delta_p w_{ij} = \eta \delta_{pi} a_{pj}$$

$$\Delta_p w_{20} = \eta \delta_{p2} a_{p0} = 0.2 * (-0.002972) * (0.017322) = -0.000010$$
$$w_{20}^{new} = w_{20}^{old} + \Delta_p w_{20} = 0.121845 + (-0.000010) = 0.121835$$

$$\Delta_p w_{62} = \eta \delta_{p6} a_{p2} = 0.2 * (0.021013) * (0.266492) = 0.001120$$

$$\Delta_p b_2 = \eta \delta_{p2} = 0.2 * (-0.002972) = -0.000594$$
$$b_2^{new} = b_2^{0ld} + \Delta_p b_2 = (-0.444700) + (-0.000594) = -0.4452294$$

# The Back-propagation Neural Networks

— A multiplayer feed-forward net trained by back-propagation algorithm.

— The largest drawback: convergence time.

## Activation function

An activation function for a back-propagation net should have several important characteristics:

1. Continuous
2. Differentiable
3. Monotonically non-decreasing
4. Easy to compute

## Some kinds of activation function:

| | Binary sigmoid | Bipolar sigmoid | Hyperbolic sigmoid |
|---|---|---|---|
| Type of function | $\frac{1}{1+e^{-x}}$ | $\frac{2}{1+e^{-x}}-1$ | $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ |
| Range of output | (0 ,1) | (-1,1) | (-1,1) |

## How many training pairs there should be?

$P$ : the number of training patterns available.
$W$ : the number of weights to be trained.
$e$ : the accuracy of classification expected.

$$e = \frac{W}{P} \qquad P = \frac{W}{e}$$

For example, if one wants an accuracy level of at least 90%, corresponding to $e = 0.1$, if W=80, then one should use 800 training patterns.

Baum, E. B. and Hausler, D., 1989, “What Size Net Gives Valid Generalization?” *Neural Computation*, Vol. 1, pp, 151-160.

## Number of hidden layers

— One hidden layer is sufficient for a back-propagation net to approximate any continuous mapping from the input patterns to the output patterns to an arbitrary degree of accuracy.

— A second hidden layer may provide improved performance when the mapping is particularly complex or irregular. The need for more than two hidden layers is highly unlikely.

— In general, one should keep the net as simple as possible.

## Number of units

—If a network has too few hidden units, it cannot learn the training set well. On the other hand, networks with too many hidden units tend to memorize the training set (i.e., over-training) and cannot generalize well.

— One can use the rule-of thumb that the number $h$ of $PE_S$ in the first hidden layer should be about $h = \dfrac{P}{10(m+n)}$ , where $P$ is the number of training patterns and $n$ and $m$ are the number inputs and outputs, respectively.

Widow, B., Winter, R. G. and Baxter, R. A., 1987, "Learning phenomena is Layered Neural Networks," *Proceeding of the First IEEE International Conference on Neural Networks*, San Diego.

— When two hidden layers are used, the number of $PE_S$ will be less than the number used in the first hidden layer, say about one half as many.

Note: Back-propagation networks do not have to be fully interconnected, although most applications work has been with fully interconnected layers.

## Neural training

When the back-propagation neural network is applied, the large the learning rate, the large changes in the weight ($\Delta_p w_{ij} = \eta\delta_{pi}a_{pj}$). However, a large learning rate may lead to oscillation. One way to increase the learning rate without leading to oscillation is to modify the back-propagation learning rate according to the following equation:

$$\Delta_p w_{ij}^{new} = \eta\delta_{pi}a_{pj} + \alpha w_{ij}^{old}$$

where $\alpha$ is a constant (momentum coefficient) that determines the effect of past weight changes on the current direction of movement in weight space. $\eta$ and $\alpha$ are chosen by the neural network user.

Based on the experimental experience, the value of $\eta$ can be set a low value (about 0.10 to 0.30) in the beginning. By gradually increasing the value of $\eta$, one can find a suitable learning rate. On the other hand, the value of $\alpha$ can be set a high value (about 0.80 to 0.95) in the beginning. By gradually decreasing the value of $\alpha$, one can find a proper momentum coefficient.

When the learning rate and momentum coefficient are determined, by choosing the feasible number of layers and number of nodes in the hidden layers, the back-propagation network can be used effectively in estimating the mapping function. The value of root mean squared (RMS) error for all patterns can be used as an index for the performance of the trained network. If RMS reaches a stable condition (or is less than some criterion), the network stops training.

The amount of training and the order of pattern presentation have a strong influencer on the network performance. For most applications, pattern presentation should be randomly sequenced.

## How long to train the net?

- Hecht Nielsen suggests using two sets of data during training:

  A set of training patterns

  A set of training-testing (testing) patterns

- Since the RMSE will never actually reach zero, it is best to decide in advance a tolerable RMSE value for witch training can be stopped or train the network until the difference of the RMSE of two successive iterations is less than a pre-determined tolerance.

## Cross-validation

— The available data set is randomly partitioned into a training set and a test set.

— The training set is further partitioned into two disjoint subset:

- Estimation subset, used to select the model.
- Validation subset, used to test or validate the model.

— The generalization performance of the selected model is measured on the test set.

## K-fold validation:

## Over-training

— If the same training patterns are given to the neural network over and over, and the weights are adjusted to match the desired outputs, we are essentially telling the network to memorize the patterns, rather than to extract the essence of the relationships. (Bigus, 1996)

— Over-training can usually be avoided by adhering to the following rules:

− Selecting the proper number of hidden-layer PEs for the network.

− Choosing a sufficiently large training set when possible.

− Selecting and presenting the patterns randomly during the training process.

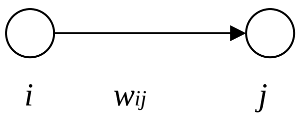
− Using a combination of the above methods.

## Pruning a network

Pruning a network is the process of removing unnecessary PEs and connections (weights).

Procedure for pruning include the following:

1. When unit $i$ has $a \pm d$ over the whole training set with $a$ and $d$ constant and $d$ a small value, then, for each unit $j$ in the next layer connected to unit $i$, substitute for the bias input of unit $j$, $w_{bias}$, the new bias weight $w'_{bias} = w_{bias} + aw_{ij}$ and remove unit $i$. The resultant network is essentially unchanged.



2. When units $p$ and $q$ have almost the same output over the training set, one (say $p$) can be removed. At each unit $j$ in the next layer that $p$ is connected to replace weight $w_{qj}$ to unit $q$ with $w'_{qj} = w_{pj} + w_{qj}$. The resultant network is essentially unchanged.

3. Finally, set all weights with relatively small values equal to zero. For example, weight values that are less than 10% of the average absolute magnitude of the other weights can often be eliminated with little or no effect on the network’s performance.

# Kohonen Self-Organizing Maps

## Kohonen's SOM

-- Unsupervised learning.

-- A topological map.

-- A two-layered network (the two layers are fully interconnected).

-- Usually the second layer is organized as a two-dimensional grid.

-- Random values may be assigned for the initial weights.

-- SOM can be used to cluster a set of continuous-valued vectors **x** = ($x_1$, $x_2$, ..., $x_n$) into *m* clusters.

Input: **x**

Weights: $\mathbf{w}_j$

The matching value for each unit *j* is $\|\mathbf{x} - \mathbf{w}_j\|$.

The unit with the lowest matching value (the best match) wins the competition.

## The operation of a Kohonen network

1. Compute a matching value for each unit in the competitive layer.

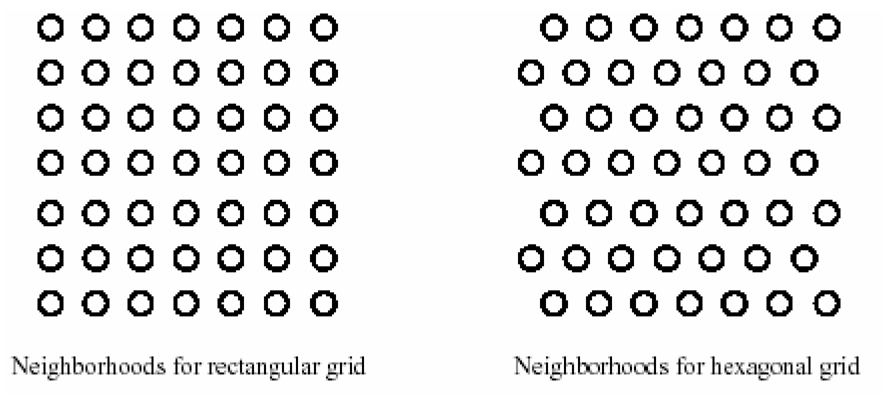
2. Find the best match.

3. Identify the neighborhood around the winner unit.

4. Weights are updated for all units are in the neighborhood of the winning unit.

$$\Delta w_{ij} = \begin{cases} \eta(x_i - w_{ij}) & \text{if unit } j \text{ is in the neighborhood } N_c \\ 0 & \text{otherwise} \end{cases}$$

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$


Neighborhoods for rectangular grid
Neighborhoods for hexagonal grid

There are two parameters that must be specified:

1. The value of the learning rate $\eta$.
2. The size of the neighborhood $N_c$.

Example:

$$\eta_0 = 0.2 \sim 0.5$$

$$\eta_t = \eta_0 (1 - \frac{t}{T})$$

$$R_t = R_0 (1 - \frac{t}{T})$$

$t$: the current training iteration

*T:* the total number of training iterations to be done.

Typical values for $R_0$ may be chosen at a half or a third of the width of the competitive layer of processing units.

Note: The learning rate $\eta$ is a slowly decreasing function of time (or training epochs).

The radius of the neighborhood around a cluster unit also decrease as the clustering process progresses.

Algorithm (Fausett, 1994)

```
Step 1: Initialize weights $w_{ij}$
    Set topological neighborhood and learning rate parameters.
Step 2: While stopping condition is false, do Steps 3-9.
    Step 3: For each input vector **x**, do Steps 4-6.
        Step 4: For each $j$, compute:
            $D(j) = \sum_i (w_{ij} - x_i)^2$
        Step 5: Find index $j$ such that $D(j)$ is a minimum.
        Step 6: For all unit $j$ within a specified neighborhood of $j$, and for
            all $i$:
            $w_{ij}^{new} = w_{ij}^{old} + \eta(x_i - w_{ij}^{old})$
    Step 7: Update learning rate.
    Step 8: Reduce radius of topological neighborhood at specified times.
    Step 9: Test stopping condition.
```

## Simple example

For example, A SOM to cluster four vectors

Let the vector to be clustered be

(1,1,0,0); (0,0,0,1); (1,0,0,0) and (0,0,1,1).

The maximum number of cluster to be formed is $m = 2$.

Suppose the learning is $\eta(0) = 0.6, \eta(\mathrm{t}+1) = 0.5\eta(\mathrm{t})$.

Only the winning unit is allowed to learn, i.e., $R = 0$.

Step 1: initial weight matrix:

$$\begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

Initial radius: $R = 0$. Initial learning rate: $\eta(0) = 0.6$.

For the first vector (1,1,0,0),

$$D(1) = (0.2-1)^2 + (0.6-1)^2 + (0.5-0)^2 + (0.9-0)^2 = 1.86;$$
$$D(2) = (0.8-1)^2 + (0.4-1)^2 + (0.7-0)^2 + (0.3-0)^2 = 0.98.$$

The input vector is closest to output node 2, so $j = 2$.

The weights on the winning unit are updated:

$$w_{i2}^{new} = w_{i2}^{old} + \eta(0)(x_i - w_{i2}^{old})$$

$$= \begin{bmatrix} 0.8 \\ 0.4 \\ 0.7 \\ 0.3 \end{bmatrix} + (0.6)\begin{bmatrix} 0.2 \\ 0.6 \\ -0.7 \\ -0.3 \end{bmatrix} = \begin{bmatrix} 0.92 \\ 0.76 \\ 0.28 \\ 0.12 \end{bmatrix}$$

This gives the weight matrix:

$$\begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 0.5 \\ 0.5 & 0.0 \\ 1.0 & 0.0 \end{bmatrix}$$